ANL/MCS-TM-343

Mesh Copy/Move/Merge Tool for Reactor Simulation Applications

Mathematics and Computer Science Division

ANL/MCS-TM-343

Mesh Copy/Move/Merge Tool for Reactor Simulation Applications

prepared by
Timothy J. Tautges and Rajeev Jain
Mathematics and Computer Science Division, Argonne National Laboratory

prepared for
U.S. Department of Energy Reactor Campaign

April 30, 2010

EXECUTIVE SUMMARY

Reactor core simulations require the construction and mesh generation for core models consisting of lattices of fuel and other rods grouped into assemblies, and lattices of assemblies of several types grouped into a core model. A set of tools has been described for generating assembly and core lattice models. Both rectangular and hexagonal lattices are supported. The tools operate in three stages. First, assembly models of various types can be generated by the AssyGen tool, based on input describing the content of unit cells, the arrangement of unit cells in the lattice, and the extent of the lattice and any surrounding material. After generating the assembly model, the model is meshed with the CUBIT mesh generation toolkit, optionally based on a journal file output by AssyGen. After one or more assembly model meshes have been constructed, they are arranged in a core model using the CoreGen tool. The input for CoreGen is similar to that of AssyGen, with assembly models substituted for unit cells. AssyGen and CoreGen also annotate the models with material and volume groupings necessary for specifying materials and boundary conditions required by the analysis. The AssyGen and CoreGen tools are packaged in the open-source MeshKit library for mesh generation; download and build instructions are included in this document.

Various improvements and enhancements for the AssyGen and CoreGen tools are described. In the area of automation, improved meshing of assembly top surfaces and automated checking of the divisions on the sides of assemblies to guarantee a contiguous core model mesh are suggested. Suggested enhancements include elimination of the requirement for pins to be inside unit cells if CellMaterial cards are not used, and development of a visualization capability for viewing models generated by CoreGen.

Table of Contents

Table of Figures

MESH COPY/MOVE/MERGE TOOL FOR REACTOR SIMULATION APPLICATIONS

1. Introduction

Geometry and Mesh generation are important parts of reactor simulation, since they account for a large part of the interactive time required for these simulations. Reactor core geometries are also characterized by lattices of repeated structures, both for individual assemblies (formed by lattices of fuel pins) and for the overall core (formed by lattices of assemblies). A process consisting of several tools has been developed to generate geometry and mesh for reactor core models, taking advantage of the lattice-based structure in these models. Based on a small set of parameters, geometry and mesh are constructed for a variety of reactor core types arranged as both square and hexagonal lattices. The output of these tools can be used as-is, or can be further customized by users to adjust local meshing parameters. Example assembly and core models generated using this process are shown in Figure 1. Neutron transport computations with the UNIC code have been performed on models generated by this process.

The tools described in this document rely on codes and libraries from a variety of sources. Geometric models are constructed through the iGeom Application Programming Interface (API), implemented using the Common Geometry Module (CGM) [1]; we refer to this combination of interface and implementation as iGeom/CGM. iGeom/CGM provides functions for constructing, modifying, and querying geometric models in solid model-based and other formats. For this effort, the primary modeling engine used is ACIS [2], accessed through the CUBIT mesh generation tool [3]. Finite element mesh models are constructed and stored through the iMesh API [4], implemented by the MOAB library. iMesh/MOAB provides query, construction, and modification of finite element meshes, plus polygons and polyhedra. The CUBIT mesh generation toolkit is a code for generating finite element meshes of both hexahedral and tetrahedral elements [2]. CUBIT provides automated meshing for the lattice-type geometric models discussed here. Because CUBIT is a closed-source code, efforts are being made to support lattice-type mesh generation with MeshKit, an new open-source mesh generation library being developed at Argonne.

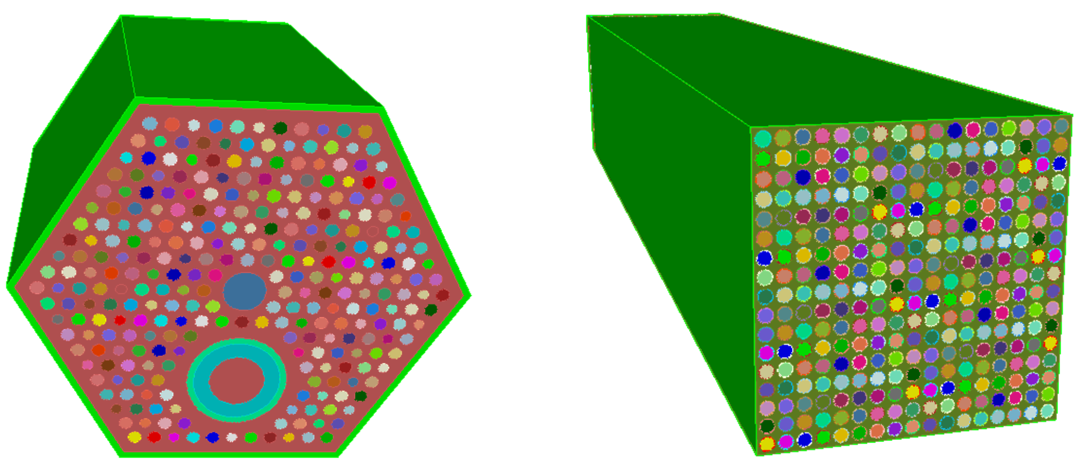

Figure 1: Hexagonal (left) and cartesian (right) assembly models produced with AssyGen.

The overall workflow required to construct reactor assembly and core geometry and meshes requires execution of several steps. First, one or more reactor pin assembly geometric models are constructed, based on user input specifying number and location of cylindrical pins and surrounding duct wall. The output of this first step is both a geometric model, and a template journal file, which can be used to mesh the assembly model using CUBIT. Next, meshes are generated with CUBIT for each assembly type, using the generated journal file and any user modifications to that meshing procedure. Finally, a core model is generated by reading individual assembly mesh types, and copy/moving them into a lattice arrangement specified by the user. This process also outputs a "makefile", which helps automate regeneration of assembly and core meshes after modifications to the various input files. This multi-step process provides both an automated procedure for generating assembly and core meshes, as well as a method for user intervention in specific parts of this construction process without requiring modification of the majority of the overall construction approach and input.

This report is organized as follows. In the process described here, assembly geometry and core meshes are constructed using the AssemblyGen and CoreGen tools, respectively. Section 2 describes the overall workflow of this system, and the coordination of their execution and the associated input/output files. Section 3 describes the input required by the AssemblyGen tool, and how the resulting geometric models are meshed using CUBIT. Section 4 describes the use of CoreGen to generate reactor core models based on those assembly meshes. Section 5 gives various examples of geometry and mesh models generated using this process, as well as an assembly neutron transport calculation based on one such model. Conclusions and future directions are given in Section 6. Instructions for downloading, building, and installing the tools described in this document appear in Appendix A. Complete input file listings appear in Appendix B.

2. Overall workflow

The process for constructing assembly geometry and core mesh models requires execution of several tools, using input files generated automatically or based on user input. The overall process workflow, including the codes run and the input/output files used, is shown in Figure 2. This discussion refers to the AssemblyGen and CoreGen tools by reference; more detailed descriptions of these tools is given in Sections 3 and 4, respectively. Section 3 also includes a description of assembly mesh generation using the CUBIT mesh generation toolkit.

In the first stage of execution, the user constructs one input file for each type of assembly type, vx.inp (x = 1, 2, …). This file describes the assembly as a lattice of cylindrical pins, annular regions surrounding each pin (e.g. for pin cladding), and overall assembly dimensions and duct wall characteristics. AssemblyGen reads this file and produces three files, a geometric model (by default stored in ACIS format in a file with the ".sat" extension) and a pair of journal files describing a default method for generating a mesh for the assembly.

In the second stage of execution, the user generates a mesh for each assembly type. In its simplest form, this stage requires specification of a mesh size and execution of CUBIT using the vx.jou journal file produced by AssemblyGen. The resulting mesh is stored in CUBIT's save/restore format, in a file vx.cub. If desired, the user can modify vx.jou to tailor the mesh generation to their individual needs, or substitute their own version of this file for generating the assembly mesh. The only explicit requirement on the resulting mesh is that the mesh on outer boundaries of the assembly match the boundaries of neighboring assemblies, or of the interstices material in cases where assemblies do not completely fill the core space.

In the final stage, an input file core.inp is processed by CoreGen, which specifies the names of assembly mesh files and their placement in an overall core model. This model is output in MOAB's native ".h5m" format, and includes the various material and boundary condition groups propagating through the process from the assembly input files.

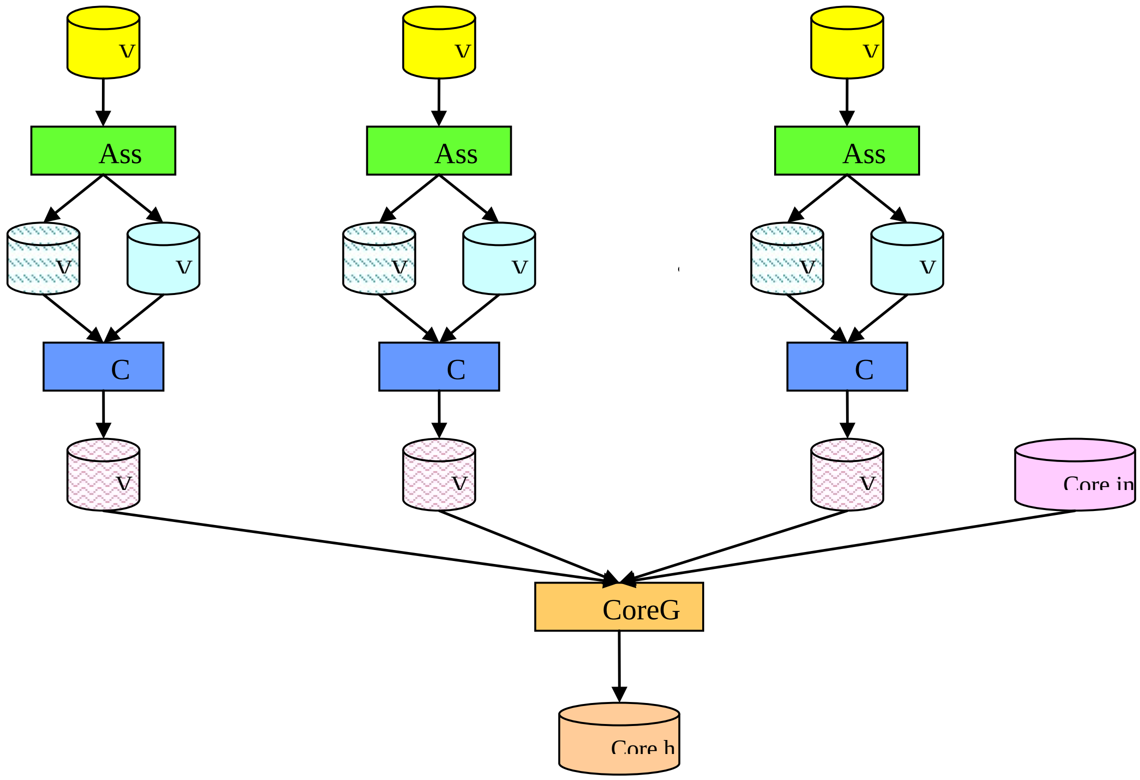

Figure 2: Overall workflow for generating reactor models.

3. Generating Assembly Geometric Models with AssyGen

A reactor core assembly is a lattice of unit cells, where each unit cell has zero or more concentric cylindrical layers of material, cut from a background material. An assembly can be surrounded by one or more layers of duct wall material. The lattice can be either hexagonal or rectangular. Hexagonal lattices are described by the number of rings starting from a center cell, with hexagonal unit cells arranged such that their centers are separated by a measure called the assembly pitch. The total number of unit cells in a hexagonal lattice of N rings is 3*N*(N-1). Rectangular lattices are characterized by the number of unit cells in the X and Y directions, with centers separated by the same pitch. Figure 1 shows examples of hexagonal and rectangular assemblies.

The AssyGen tool constructs hexagonal and rectangular assemblies based on an input file specified by the user. The input file syntax for AssyGen is shown in Figure 3. Keywords in this file are as follows:

1. GeometryType: This card defines the geometry type and can take values ‘Hexagonal’ or ‘Cartesian’ depending on the type of assembly specified in the ‘Assembly’ card
2. Materials: Materials card defines all the material names followed by their aliases. First input to this card is the number of materials. Material aliases defined here are used in subsequent cards to assign materials.
3. Dimensions: The outer covering of the assembly is specified using this card. The first input specifies the number of coverings. Next input is X, Y location of center of the covering, this is followed by the Z start and Zend locations for all the coverings. The order in which the input is specified depends on the value of GeometryType specified. Hexagonal geometry will require only one dimension (Pitch) whereas Cartesian geometry will require two dimensions (Length and Width). Finally, material alias for the coverings are specified.

4. Pincells: This card's first input is the number of pincells. When GeometryType is 'Hexagonal' and all the pincells have the same pitch. The pitch value is distance between the flats of a hexagon surrounding that encloses the pincell. The next input in the pincell cards is the pitch value. 'Cartesian' geometry has two pitch values X and Y. Alternatively; all pincells can define their pitch using 'Pitch' Card.
5. 'Pincell Alias': This is not a card, it begins with pincell name defined by the user, followed by alias and finally the number of subsequent lines this pincell would use to define its properties. The program loops through these lines to create this pincell. For describing a pincell 3 cards can be used: Pitch, Cylinder and CellMaterial.
6. Pitch: This card does not appear in the example described above, since the pitch is described in the 'Pincells' card for all the pincells. In Fig.2 corresponding to test input file: TwoPin.inp (see Appendix I) this card is used. For 'Hexagonal' Geometry this card takes only one value which is the pitch of the hexagon. For 'Cartesian' geometry this cards takes 3 inputs – length, width and height of the pincell.
7. Cylinder: First input to this command is the number of concentric cylinders in the pincell. Next X, Y of the center and the Z start and Z end of the cylinders is specified. Finally the radii and material alias for the cylinders are specified. This card is Optional.
8. CellMaterial: This creates a cell (covering) outside the cylinder. Input specification is similar to Cylinder card. For a 'Hexagonal' this command creates a hexagon outside the cylinders and a square in case of 'Cartesian' Geometry. This card is optional.
9. Assembly: Once all the pincells are defined. The assembly arrangement is specified using this card. Input to this card also depends on the GeometryType card. It takes 2 values for 'Cartesian' that is no. of pincells in X, Y. For 'Hexagonal' geometry type it takes one input as the number of pincells on one side of the hexagon. Next line onwards the aliases of the pincells are arranged in the desired form to specify the assembly.
10. Rotate: Rotates the entire assembly model by the specified angle (in degrees) around the specified axis.
11. Center: Centers the entire assembly model about the origin.
12. Section: Subtracts everything on the negative side of the specified axis or, if Reverse is given, the positive side of that axis. If offset is given, model is sectioned at that coordinate on the specified axis, instead of at the zero value.
13. AxialMeshSize: Mesh size in the axial direction; this size is written to the journal file for meshing the assembly model.
14. RadialMeshSize: Mesh size in the radial direction; this size is written to the journal file for meshing the assembly model.
15. End: This commands marks the end of input file.

Examples of AssyGen input are discussed in Section 6, for both the hexagonal and rectangular assemblies shown in Figure 1.

AssyGen constructs a geometric model, annotated with the following names:

Volumes/Regions: Volumes are named according to their material type, which can be used to assign material properties.

Faces/Surfaces: Surfaces at the top, bottom, and sides of the assembly are named using "_top", "_bot", and "_side" suffixes. These names can be used to group surfaces for the application of boundary conditions. For example, for an assembly with upward flow, surfaces whose names end in "_bot"/"_top" can be placed in groups for application of inlet/outlet boundary conditions, respectively. Surfaces can be filtered by containing volume using set booleans, e.g. to select fluid regions outside of cylindrical fuel pins.

The model is output to a file with the same base name as the input file, with a ".sat" (ACIS) or ".brep" (OpenCascade) extension.

4. Mesh Generation using CUBIT

In addition to generating the geometric model, AssyGen also writes two journal files, <base_name>.jou and "template.jou". The first of these contains the basic commands necessary to mesh the assembly model, define various material groups and boundary condition sets, and output the results to <base_name>.cub. template.jou defines several Aprepros parameters used in the assembly meshing process, and is meant to be shared by several assembly journal files; parameters are defined for mesh sizes and various user-selectable mesh schemes. In many cases, the user will only need to specify a mesh size in template.jou, after which the meshing process for an assembly will be completely automatic. In other cases, e.g. if a relatively coarse mesh size is requested or finer-grained control over mesh density or quality is required, users can modify <base_name>.jou as necessary before running CUBIT on that file. Once the journal file has been adjusted and the mesh for the assembly generated, users generate any other assembly meshes required, then proceed to the next stage, described in the next section.

Sample journal files produced by AssyGen for the models in Figure1 are given in Section 6.

5. Generating Core Model with CoreGen

A reactor core is formed by placing various assembly types in a lattice, possibly surrounded by material in the interstices regions. CoreGen is used to generate core models, using syntax similar to that used in AssyGen.

```
GeometryType {Hexagonal | Cartesian}
Materials <nmat>  {<material name>  <material alias>} * nmat
Duct <n_layers> <x_center> <y_center> <z_start> <z_end>
  {<duct_rad> (Hexagonal) | <duct_x> <duct_y> (Cartesian) } * n_layers
  {material} * n_layers
Pincells <n_cells> <pitch>
{
  <cell_name>  <cell_alias>  <n_line>
  {Cylinder <n_cyl> <x_0> <y_0> <z_start> <z_end> {<r_cyl>} * n_cyl
            {mat}*n_cyl} * n_line} * n_cell
Lattice {n_ring | n_x  n_y}
{
  {<cell_alias>} * (3*n_ring*(n_ring-1))  |  ! GeometryType=Hexagonal
  {<cell_alias>} * n_x*n_y                             ! GeometryType=Cartesian
}
[Rotate {x | y | z} <angle>]
[Center]
[Section {x | y | z}  <offset>  [reverse]  ]
[Move <x> <y> <z> ]
[AxialMeshSize <size>]
[RadialMeshSize <size>]
END
```

Figure 3: Syntax used with AssyGen.

The syntax used as input to CoreGen is shown in Figure 4. Similar to the input for AssyGen, a GeometryType is specified as hexagonal or Cartesian. Assemblies specifies the number of different assembly types, and the pitch of the lattice arrangement. For each assembly type, a file name containing the mesh for that assembly type and the alias used to reference that assembly type are given. The Lattice keyword specifies the number of rings (hexagonal) or the number of assemblies in the X and Y directions (Cartesian) for the core. Following this, assembly type aliases are listed in the order they appear in the core, similar to how the lattice is specified for AssyGen. The Symmetry keyword specifies an integer number of pieces which would form an entire core; for example, n_sym=6 indicates the input is for 1/6 of a core. For hexagonal lattices, the model starts at the x=0 axis and ends at an angle of 2□/n_sym; allowed values for n_sym are 2, 3, and 6. For rectangular lattices, allowed values are 2 and 4, denoting ½ and ¼ symmetry starting in the +x, +y quadrant. If the Background keyword appears, the specified assembly mesh is also imported, without being moved relative to the core lattice. This is used to represent material in interstices regions between assemblies, along with any structure surrounding the core, like a core barrel.

```
Geometry {Surface | Volume}
GeometryType {Hexagonal | Cartesian}
Assemblies <n_assys> {<pitch_x> <pitch_y> (if GeometryType=Cartesian) |
                      <pitch> (if GeometryType=Hexagonal}
{<mesh_file> <assy_alias>} * n_assys
Lattice {<n_x> <n_y> (if GeometryType = Cartesian) |
         <n_rings> (if GeometryType = Hexagonal)}
{<assy_alias>} * {<n_x>*<n_y> (if GeometryType = Cartesian) |
                  3*<n_rings>*(<n_rings>-1) (if GeometryType = Hexagonal)}
END
```

Figure 4: Syntax used with CoreGen.

It is important to note here that CoreGen operates on mesh, and makes no use of the geometric models produced by AssyGen; those models are for the sole purpose of generating the mesh models.

After copy/moving the assembly meshes into the core lattice, coincident nodes are merged, resulting in a contiguous mesh.

6. Examples

There are several examples in this section. First, examples of hexagonal and Cartesian assemblies are given, with each assembly being relatively simple. For the hexagonal assembly case, a UNIC neutronics simulation is demonstrated on that model. Next, an example takes the hexagonal assembly from earlier, along with a second assembly, and constructs a core lattice from them. Finally, a larger example is given which constructs a core model from X different assembly types. The primary input files used for all these examples are included here; other input files used in the process are included in Appendix X. These examples are also part of the AssyGen and CoreGen code distribution, and are used in unit testing of the capability.

6.1 Single Hexagonal Assembly

This example shows the construction of a single VHTR assembly using AssyGen. The input for this example is shown in Figure 4. There are 6 pin cell types, with three sizes of pins representing fuel, burnable poison, and coolant regions; a structure at the center describes a fuel handling hole and a larger

```
! Hexagonal VHTR Control Fuel Block Assembly Definition
! without CellMaterials (Prism) around individual Pins
!Geometry Surface
GeometryType Hexagonal
Materials 10 Mat_C2Coolant C02 Mat_C1Coolant C01 Mat_OCoolant C1Out Mat_Coolant C1
   Mat_SCoolant SC1 Mat_Fuel  F1 Mat_Block G1 Mat_Poison P1 Mat_Control R1 Mat_FHH
H1
Dimensions 2 0.0 0.0 -0 50.0 36.0 37.3  G1 C1Out
Pincells 6 1.87960
! 1
Cell_Fuel       FC 1
Cylinder  1 0.0 0.0    0.0 50.0   0.6225  F1
! 2
Cell_BurnPoison BP 1
Cylinder  1 0.0 0.0    0.0 50.0   0.6225  P1
! 3
Cell_SmallCoolant    SC 1
Cylinder  1 0.0 0.0    0.0 50.0   0.6350 SC1
! 4
Cell_Coolant    CC 1
Cylinder  1 0.0 0.0    0.0 50.0   0.5075 C1
! 5
Cell_Control CR 1
Cylinder  3  0.0 0.0 0.0 50.0 2.64 4.13 4.7625 C01 R1 C02
! 6
Cell_FHH     FH2 1
Cylinder  1  0.0 0.0 0.0 50.0 2.0 H1
Assembly 11
                     BP  FC  CC  FC  FC  CC  FC  FC  CC  FC  BP
                   FC  CC  FC  FC  X   XX  XX  XX  FC  FC  CC  FC
                 CC  FC  FC  CC  XX  XX  XX  XX  XX  CC  FC  FC  CC
               FC  FC  CC  FC  XX  XX  XX  XX  XX  XX  FC  CC  FC  FC
             FC  CC  FC  FC  XX  XX  XX  CR  XX  XX  XX  FC  FC  CC  FC
           CC  FC  FC  CC  FC  XX  XX  XX  XX  XX  XX  FC  CC  FC  FC  CC
         FC  FC  CC  FC  FC  CC  XX  XX  XX  XX  XX  CC  FC  FC  CC  FC  FC
       FC  CC  FC  FC  CC  FC  FC  XX  XX  XX  XX  FC  FC  CC  FC  FC  CC  FC
     CC  FC  FC  CC  FC  FC  CC  FC  FC  SC  FC  FC  CC  FC  FC  CC  FC  FC  CC
   FC  FC  CC  FC  FC  CC  FC  FC  SC  XX  XX  SC  FC  FC  CC  FC  FC  CC  FC  FC
 BP  CC  FC  FC  CC  FC  FC  CC  FC  XX  FH2 XX  FC  CC  FC  FC  CC  FC  FC  CC  BP
   FC  FC  CC  FC  FC  CC  FC  FC  SC  XX  XX  SC  FC  FC  CC  FC  FC  CC  FC  FC
     CC  FC  FC  CC  FC  FC  CC  FC  FC  SC  FC  FC  CC  FC  FC  CC  FC  FC  CC
       FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC
         FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC
           CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC
             FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC
               FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC
                 CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC
                   FC  CC  FC  FC  CC  FC  FC  CC  FC  FC  CC  FC
                     BP  FC  CC  FC  FC  CC  FC  FC  CC  FC  BP
Center
!Rotate Z 30
!Section Y 0
!Section X 0
!Rotate Z -30
!Section Y 0
END ! This is a must
```

hole in the lower half of the assembly represents a control rod and guide tube. Note that empty pin cells are specified in cell positions surrounding these larger holes, to leave room for these larger structures. The resulting geometric model is shown in Figure 6.

Figure 5: AssyGen input for simple Hexagon case.

AssyGen also outputs two CUBIT journal files, v1.jou and schemes.jou. After specifying a mesh size in schemes.jou, the mesh shown in Figure 5 (right) was generated. It is important to note that generating an all-hex mesh for this model is difficult, due to the many cylindrical pins cut out of the top surface of the block. A fine mesh can be generated without too much difficulty; however, generating a coarse mesh for this model would require a substantial effort to make targeted modifications to the assembly geometry and mesh parameters.

The model generated by this example was used as input to the UNIC neutron transport code [5]]. Figure 6 shows the material compositions used for this problem (left) and the thermal neutron flux computed (right). The flux is much lower in the region of the large control rod, and slightly lower around the six burnable poison pins (located at corners of the hexagonal assembly), as expected. Note that in this case a tetrahedral mesh was used, which was generated by modifying input to the v1.jou file output by AssyGen.

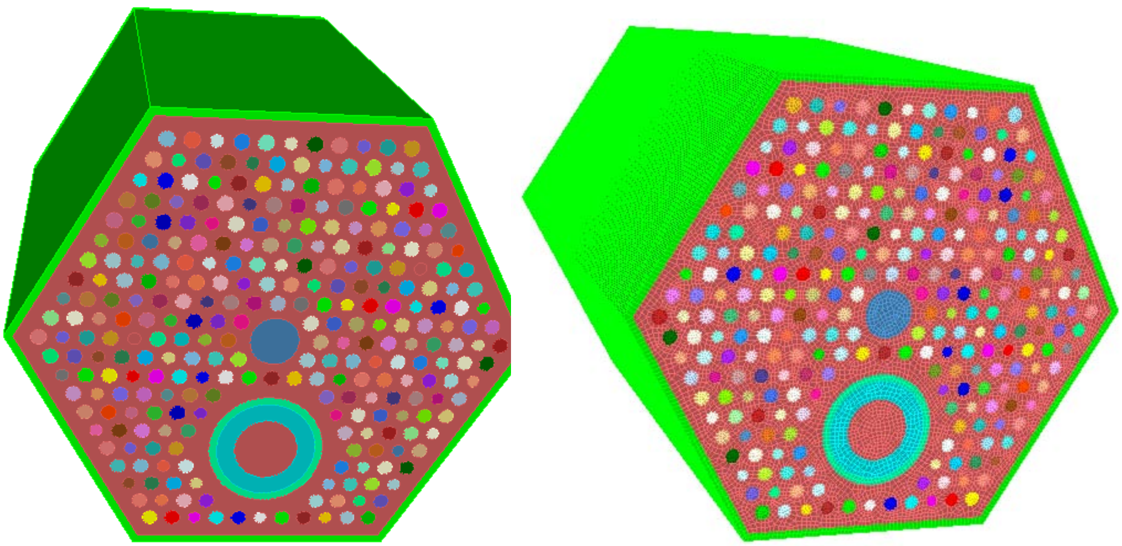

Figure 6: Hexagonal assembly constructed by AssyGen from input in Figure 4 (left); mesh generated from v1.jou output by AssyGen (right).

6.2 Simple Cartesian Assembly

In this example, an assembly with a square (Cartesian) lattice is generated. The AssyGen input for this model is shown in Figure 7, and the geometric model and mesh produced are shown in Figure 8. Fuel rods are surrounded by annular cylinders representing cladding, and the square coolant region is surrounded by another region, representing either duct wall or the space between assemblies in a core. Again, generating a good-quality mesh for this model can be difficult, depending on the element size specified by the user.

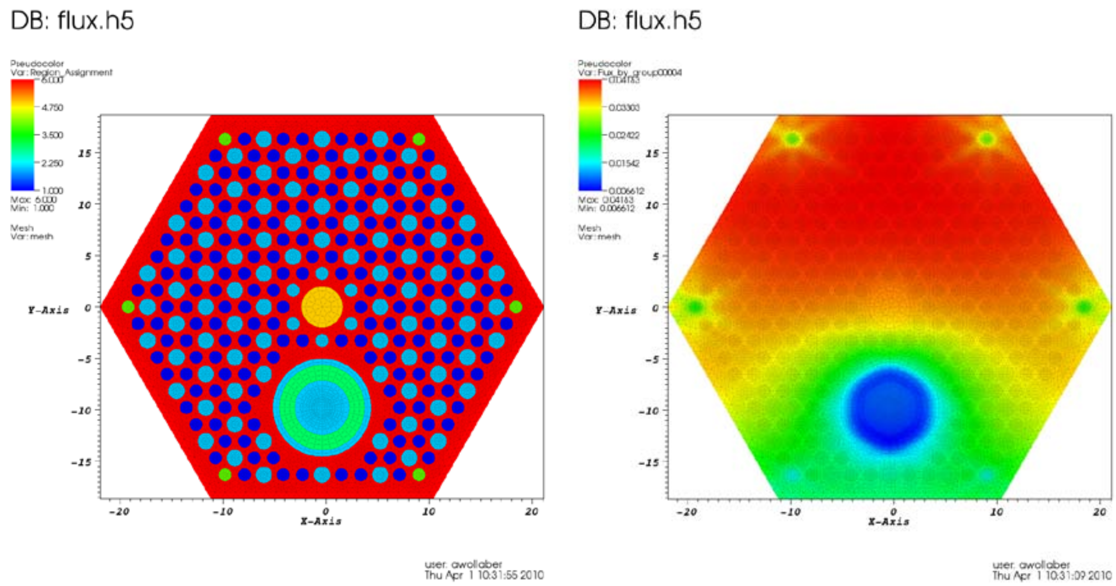

Figure 7: Material compositions used in the UNIC neutron transport calculation of the hexagonal assembly (left); computed thermal flux for the problem (right).

```
   ! ############################################################################
   ! Square PWR Assembly Definition without
   ! CellMaterials (Prism) around Pins
   ! ############################################################################
   GeometryType Cartesian
   Materials 17 PWR_FC_F_01 M1  PWR_FC_C_01 M2  PWR_FC_M_01 M3 &
   PWR_GT_F_01 M4  PWR_GT_C_01 M5  PWR_GT_M_01 M6 &
   PWR_XFC_01 M7  PWR_XFC_C_01 M8  PWR_XFC_M_01 M9 &
   PWR_CFC_01 M10  PWR_CFC_C_01 M11  PWR_CFC_M_01 M12 &
   PWR_CFC_01 M13  PWR_CFC_C_01 M14  PWR_CFC_M_01 M15 &
   Mat_Block G1 Mat_Coolant C1
   Dimensions 2 0.0 0.0 0 124.0 23.0 23.0 23.5 23.5  G1 C1 !2 outer squares at 0.0
0.0 height 0.0 to 124.0 x1 y1 x2 y2 material1 material2
   !
   Pincells 5
   ! Standard fuel cell (1)
   PWR_FC_01 FC 2
   Pitch  1.26472 1.26472 124.0
   Cylinder 2 0.0 0.0  0.0 124.0   0.5 0.6  M1 M2
   ! Guide Tube pin-cell (2)
   PWR_GT_01 GT 2
   Pitch  1.26472 1.26472 124.0
   Cylinder 2 0.0 0.0  0.0 124.0   0.5 0.6  M3 M4
   ! Left/right edge PWR cell (3)
   PWR_XFC_01 XFC 2
   Pitch  1.31472 1.26472 124.0
   Cylinder 2 0.0 0.0  0.0 124.0   0.5 0.6  M5 M6
   ! Top/bottom edge PWR cell (4)
   PWR_YFC_01 YFC 2
   Pitch  1.26472 1.31472 124.0
   Cylinder 2 0.0 0.0  0.0 124.0   0.5 0.6  M7 M8
   ! Corner edge PWR cell (5)
   PWR_CFC_01 CFC 2
   Pitch  1.31472 1.31472 124.0
   Cylinder 2 0.0 0.0  0.0 124.0   0.5 0.6  M9 M10
   Assembly 17 17
   ! 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
   CFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC CFC    !  1
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    !  2
   XFC  FC  FC  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  FC  FC XFC    !  3
   XFC  FC  FC  gt  FC  FC  FC  FC  FC  FC  FC  FC  FC  gt  FC  FC XFC    !  4
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    !  5
   XFC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC XFC    !  6
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    !  7
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    !  8
   XFC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC XFC    !  9
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    ! 10
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    ! 11
   XFC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  FC  FC XFC    ! 12
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    ! 13
   XFC  FC  FC  gt  FC  FC  FC  FC  FC  FC  FC  FC  FC  gt  FC  FC XFC    ! 14
   XFC  FC  FC  FC  FC  gt  FC  FC  gt  FC  FC  gt  FC  FC  FC  FC XFC    ! 15
   XFC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC  FC XFC    ! 16
   CFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC YFC CFC    ! 17
   END     ! This is a must
```

Figure 8: AssyGen input for Cartesian assembly.

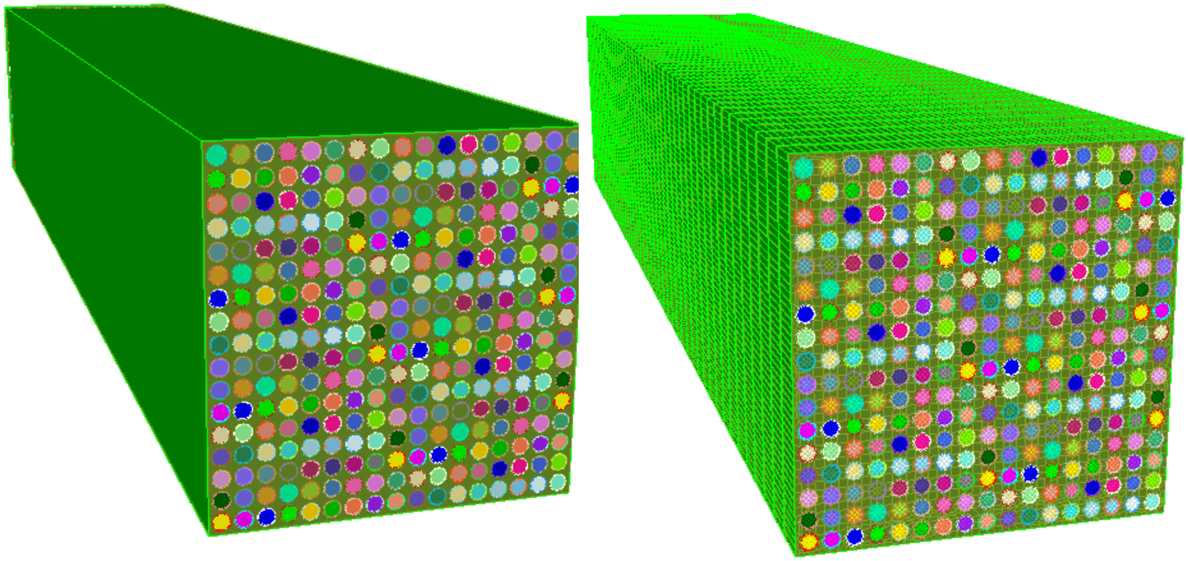

Figure 9: Geometric model produced by AssyGen input in Figure 7 (left). Mesh produced with journal file output by AssyGen (right).

6.3 Core Model with Two Assembly Types

In this example, two Cartesian assembly types are combined into a 3x3 core model. The two assemblies are shown in Figure 10. The resulting core model is shown in Figure 11.

The AssyGen input for the two assembly types is shown in Figure 12 and Figure 13. In these input files, the AxialMeshSize and RadialMeshSize keywords are used to specify mesh sizes different from the default sizes assigned by AssyGen. The CoreGen input for this problem is shown in Figure 14. This input is relatively simple for this simple core.

The makefile generated automatically by CoreGen is shown in Figure 15. Assuming AssyGen, CoreGen, and CUBIT are in the user's path, this makefile automates generation of the overall mesh; after making changes to either of the .inp files, the user can re-run 'make'. Note, though, that running AssyGen will overwrite the journal files, and CoreGen the makefile. If the user needs to make customizations to these files, they should be renamed, to avoid being over-written.

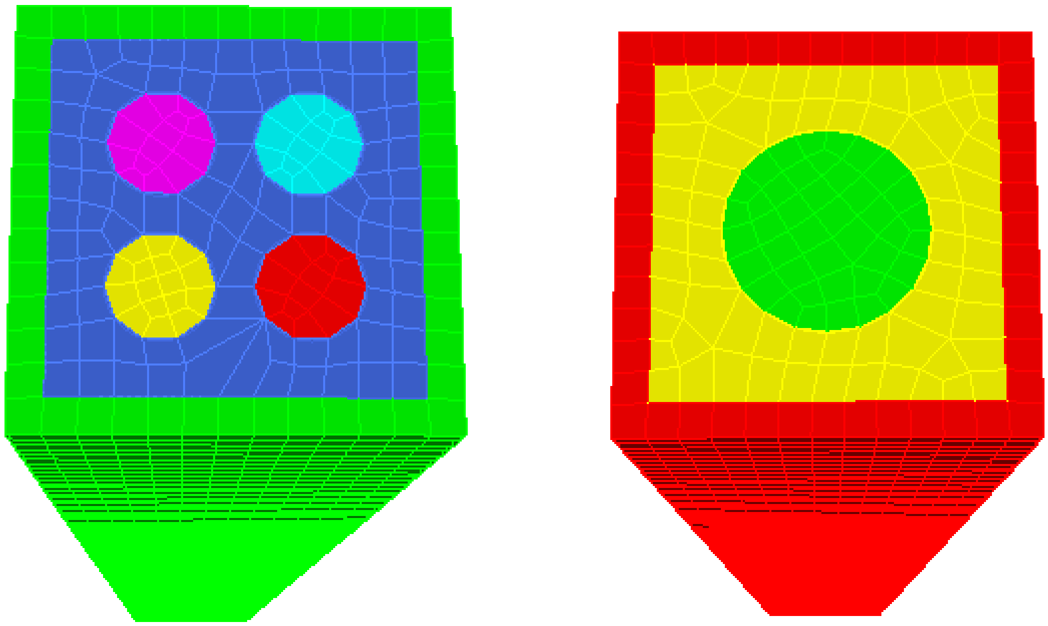

Figure 10: Two assemblies in simple 2-assembly model.

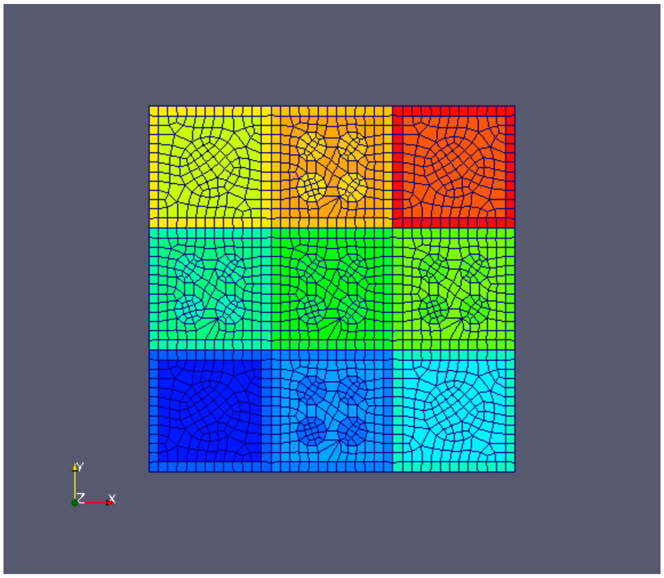

Figure 11: Core model constructed from two assembly types in 3x3 Cartesian array.

```
! ################################################################
! Square PWR Assembly Definition without CellMaterials (Prism) around Pins
! ################################################################
!
GeometryType Cartesian
!
Materials 3 PWR_FC_F_01 M1 Mat_Block G1 Mat_Coolant C1
!
Dimensions 2 0.0 0.0 0 124.0 10.0 10.0 12.0 12.0  G1 C1
Pincells 1
!
! Standard fuel cell (1)
PWR_FC_01 FC 2
Pitch  4.0 4.0 124.0
Cylinder 1 0.0 0.0  0.0 124.0   1.5  M1
!
Assembly 2 2
FC FC
FC FC
Center
AxialMeshSize 12.4
RadialMeshSize .92
END     ! This is a must
```

Figure 12: AssyGen input for first assembly type used in simple 2-assembly core.

```
! ################################################################
! Square PWR Assembly Definition without CellMaterials (Prism) around Pins
! ################################################################
!
GeometryType Cartesian
!
Materials 3 Mat_Block G1 Mat_Coolant C1 PWR_RF_R_01 M3
!
Duct 2 0.0 0.0 0 124.0 10.0 10.0 12.0 12.0  G1 C1
Pincells 1
!
! reflector
PWR_RF_01 RF 2
Pitch  8.0 8.0 124.0
Cylinder 1 0.0 0.0  0.0 124.0   3.0 M3
!
Assembly 1 1
RF
Center
axialmesh 12.4
radialmeshsize .92
END    ! This is a must
```

Figure 13: AssyGen input for second assembly type used in simple 2-assembly core.

```
! To run this test file: MeshKit> coregen 2-assms
Geometry Volume
GeometryType Cartesian
Assemblies 2 12.0 12.0
s1.cub S1
s2.cub S2
Lattice 3 3
S2 S1 S2 &
S1 S1 S1 &
S2 S1 S2
end
```

Figure 14: CoreGen input simple 2-assembly core.

```
##
## This makefile is automatically generated by coregen program
##

#specify your cubit executable's patch
CUBIT = cubit -warning -nographics -nojournal -batch -graph

COREGEN = ../../../coregen

ASSYGEN = ../../assygen

MESH_FILES = s1.cub  s2.cub

GEOM_FILES = s1.sat  s2.sat

JOU_FILES = s1.jou  s2.jou

INJOU_FILES = s1.template.jou  s2.template.jou

ASSYGEN_FILES = s1.inp  s2.inp

sq-assy.h5m : ${MESH_FILES} sq-assy.inp
   ${COREGEN} sq-assy
s1.cub : s1.sat  s1.jou  s1.template.jou
   ${CUBIT} s1.jou

s1.sat : s1.inp
   ${ASSYGEN} s1

s2.cub : s2.sat  s2.jou  s2.template.jou
   ${CUBIT} s2.jou

s2.sat : s2.inp
   ${ASSYGEN} s2
```

Figure 15: makefile for simple 2-assembly model, generated by CoreGen.

6.4 Complex Core Model

The power of using AssyGen and CoreGen is more clear when constructing models that are larger and more complex than those discussed earlier. In this example, a model is constructed for a VHTR core, with 1/6 symmetry and 11 different assembly types. The input for this model can be lengthy, and so is not repeated in this report; all files associated with this example are in the MeshKit repository, and can be browsed at http://trac.mcs.anl.gov/projects/fathom/browser/MeshKit/trunk/rgg/files/vhtr-sixth.

A few of the more interesting assembly types in this example are shown in Figure 10. The mesh in these assemblies was generated automatically, based on size and scheme input found in the template journal file. Note that one of the assemblies has a symmetry plane, so only the upper half of the assembly appears. These types of assemblies are useful for building core models with symmetry planes.

The full 1/6 core model is shown in Figure 11, along with a close-up of a region in the model. Various boundary condition groups are defined with this model.

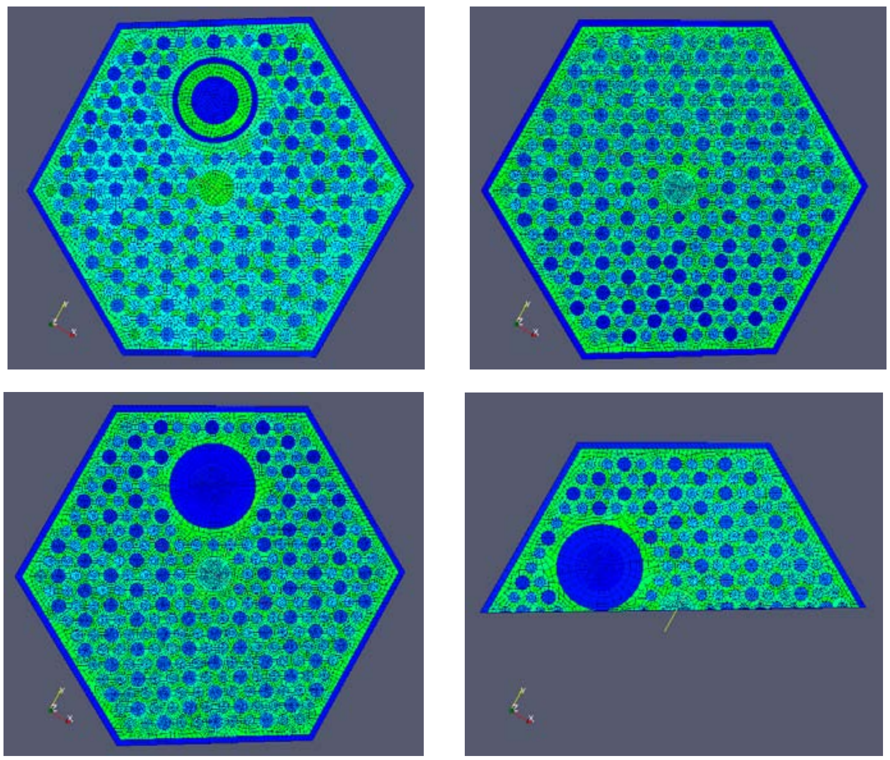

Figure 16: Four assembly types produced by AssyGen for VHTR example.

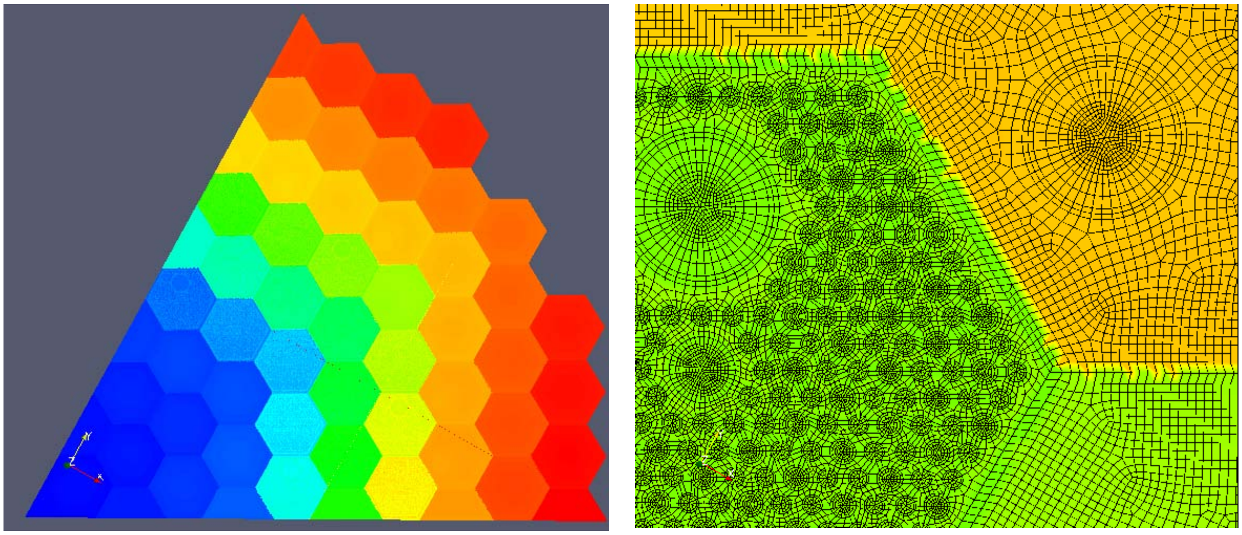

Figure 17: Full VHTR core model generated with CoreGen (left), and close-up of several assemblies (right).

7. Conclusions & Future Work

Reactor core simulations require the construction and mesh generation for core models consisting of lattices of fuel and other rods grouped into assemblies, and lattices of assemblies of several types grouped into a core model. A set of tools has been described for generating assembly and core lattice models. Both rectangular and hexagonal lattices are supported. The tools operate in three stages. First, assembly models of various types can be generated by the AssyGen tool, based on input describing the content of unit cells, the arrangement of unit cells in the lattice, and the extent of the lattice and any surrounding material. After generating the assembly model, the model is meshed with the CUBIT mesh generation toolkit, optionally based on a journal file output by AssyGen. After one or more assembly model meshes have been constructed, they are arranged in a core model using the CoreGen tool. The input for CoreGen is similar to that of AssyGen, with assembly models substituted for unit cells. AssyGen and CoreGen also annotate the models with material and volume groupings necessary for specifying materials and boundary conditions required by the analysis. The AssyGen and CoreGen tools are packaged in the open-source MeshKit library for mesh generation; download and build instructions are included in this document.

Improvements to these tools can be grouped in two areas, automation and enhanced capabilities. The process of constructing meshes for assembly geometry, while somewhat automated, has a long way to go before it will be fully automatic. This problem is manifested foremost in unstructured quadrilateral meshing of assembly top surfaces cut by large numbers of rods, such as the assemblies shown in Figure 6 and Figure 9. The test problem input for these examples contains a carefully-chosen radial mesh size, which if made larger (for a coarser mesh) will cause mesh generation for that surface to fail. This situation is quite common in mesh generation technology, where mesh generation robustness grows worse for coarser mesh sizes. This is one of the reasons for splitting the core mesh generation process into several steps, to allow user intervention mid-way through the process. For this particular problem, the solution would be either to make the mesh size finer (producing many more elements), or geometry decomposition of the assembly top surface such that the resulting smaller, less-complex surfaces are meshable with more primitive algorithms.

Another important improvement to automation could be checking the number of divisions on the "side" surfaces of assemblies being arranged in a core model, to ensure they match up and the mesh will merge into a contiguous mesh during the final stage of CoreGen. Currently, this must be checked by the user, sometimes requiring modification of mesh input.

In the area of enhancements, several specific enhancements are planned. First, AssyGen requires either that all pins in a unit cell be centered in that unit cell, or that the CellMaterial input card be used. Requiring this input line results in much more decomposition of the model on unit cell boundaries, which in turn may limit the coarseness of the mesh which can be generated for those assemblies. Methods for eliminating this requirement are being investigated.

The most difficult shortcoming in using AssyGen and CoreGen is the lack of visualization capability. This can be important for debugging the input to AssyGen, and for viewing the results of core mesh construction (assembly mesh construction can be viewed in CUBIT). This limitation will persist until a tool which can easily view the data model provided by iMesh and MeshKit is developed. Efforts are underway to develop this capability in the VisIt visualization tool [6].

Acknowledgments

This work was supported by the US Department of Energy's Scientific Discovery through Advanced Computing program under Contract DE-AC02-06CH11357. Argonne National Laboratory (ANL) is managed by UChicago Argonne LLC under Contract DE-AC02-06CH11357. This work is sponsored by the DOE AFCI program.

References

[1] T.J. Tautges, "CGM: A geometry interface for mesh generation, analysis and other applications," *Engineering with Computers*, vol. 17, 2001, pp. 299-314.
[2] *Spatial Web Site*, 2004.
[3] G.D. Sjaardema, T.J. Tautges, T.J. Wilson, S.J. Owen, T.D. Blacker, W.J. Bohnhoff, T.L. Edwards, J.R. Hipp, R.R. Lober, and S.A. Mitchell, *CUBIT mesh generation environment Volume 1: Users manual*, Sandia National Laboratories, May 1994, 1994.
[4] C. Ollivier-Gooch, L.F. Diachin, M.S. Shephard, and T. Tautges, "A language-independent api for unstructured mesh access and manipulation," *21st International Symposium on High Performance Computing Systems and Applications, HPCS 2007*, IEEE, 2007.
[5] M.A. Smith, C. Rabiti, G. Palmiotti, D. Kaushik, A. Siegel, B. Smith, T. Tautges, and W.S. Yang, "UNIC: Development of a new reactor physics analysis tool, invited," *2007 Winter Meeting on International Conference on Making the Renaissance Real*, American Nuclear Society, 2007, pp. 565-566.
[6] *VisIt User's Guide*, Lawrence Livermore National Laboratory, 2005.

A. Configure, Build, and Install Instructions

The software prerequisites for building the tools described in this document, and the means of obtaining them, are:

- CGM: the Common Geometry Module; download and build instructions can be found at http://trac.mcs.anl.gov/projects/ITAPS/wiki/CGM. Although CGM supports both the ACIS and Open.Cascade modeling engines, the latter is not yet robust enough to support AssyGen and CoreGen, so the ACIS version should be used. Note that CGM can be configured to use ACIS libraries through the CUBIT mesh generation toolkit.

- MOAB: the Mesh-Oriented database; download and build instructions can be found at http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB. MOAB depends on the HDF5 library, which is necessary for storing meshes generated by CoreGen; see MOAB build instructions for accessing and building this library.
- Lasso: a library for relating mesh to geometry. Download and build instructions are located at http://trac.mcs.anl.gov/projects/ITAPS/wiki/Lasso. Lasso requires compiled versions of MOAB and CGM before it can be compiled.
- CUBIT: a mesh generation toolkit developed at Sandia National Laboratories. See http://cubit.sandia.gov/. CUBIT is available for a nominal fee for government use in the US.

The source code for the AssyGen and CoreGen tools is released as open source software, under and LGPL license. The tools are part of the MeshKit library for mesh generation. This software resides in an svn repository located at Argonne National Laboratory. These tools are built using the following steps.

1. Download the MeshKit source code using subversion:
2. svn co https://svn.mcs.anl.gov/repos/fathom/MeshKit/trunk MeshKit
3. Change to the MeshKit source directory, and configure the library, using Linux autotools commands:
   - `cd MeshKit`
   - `autoreconf –fi`
   - `./configure –with-imesh=<MOAB_DIR> --with-cgm=<CGM_DIR> --with-lasso=<LASSO_DIR> --prefix=$PWD`

   Where <MOAB_DIR>, <CGM_DIR>, and <LASSO_DIR> are the top-level directories of the MOAB, CGM, and Lasso libraries, respectively.

4. Compile and install MeshKit:
   - `make install`
5. Build and test AssyGen and CoreGen:
   - `cd rgg`
   - `make assygen coregen`
   - `make check`

After this procedure is finished, the AssyGen and CoreGen executables will be in the <MESHKIT_DIR>/rgg directory. AssyGen and CoreGen can be run using the commands:

- `assygen <input>`
- `coregen <input>`

where <input> is the base name (the filename without the extension) of the input file. The output of AssyGen will be the files <input>.sat (the assembly model geometry), and <input>.jou and template.jou (the journal files used to generate the assembly mesh).

Any problems building, installing, or running these tools should be sent to the Fathom email list at fathom@lists.mcs.anl.gov.

TODO

1. Describe CellMaterial card

2. More detail on element blocks, sidesets in AssyGen and CUBIT syntax sections

3. Description of element blocks, sidesets in examples

4. Describe INTERSECTION card.

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439-4847

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC